Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network

Hiroki Matsutani, *Student Member*, *IEEE*, Michihiro Koibuchi, *Member*, *IEEE*, Yutaka Yamada, D. Frank Hsu, *Senior Member*, *IEEE*, and Hideharu Amano, *Member*, *IEEE*

Abstract—The topological explorations of on-chip networks are important for efficiently using their enormous wire resources for low-latency and high-throughput communications using a modest silicon budget. In this paper, we propose a novel tree-based interconnection network called Fat H-Tree that meets these requirements. A Fat H-Tree provides a torus structure by combining two folded H-Tree networks and is an attractive alternative to tree-based networks such as the Fat Trees in a microarchitecture domain. We introduce its chip layout schemes based on a folding technique for 2D and 3D ICs. Three deadlock-free routing schemes are proposed for Fat H-Tree. We evaluate the performance of Fat H-Tree and other tree-based networks using real application traces. In addition, the network logic area, wire resource, and energy consumption of Fat H-Tree are compared with other topologies, based on a typical implementation of on-chip routers synthesized with a 90-nm standard cell library. The results show that 1) a Fat H-Tree outperforms a Fat Tree with two upward and four downward connections in terms of the throughput and average hop count, 2) a Fat H-Tree requires 19.8 percent-27.8 percent smaller network logic area than the Fat Tree, 3) a Fat H-Tree consumes slightly less energy than the Fat Tree does, and 4) a Fat H-Tree uses slightly more wire resources than the Fat Tree, but the current process technology can provide sufficient wire resources for implementing Fat-H-Tree-based on-chip networks.

Index Terms—Interconnection networks, on-chip networks, network topology, tree, routing algorithm.

1 Introduction

The advances made in semiconductor technology have allowed us to integrate a number of processing cores on a single chip, and various types of Network-on-Chip (NoC) technology have been studied in order to connect them by introducing a network structure that is similar to that in parallel computers [1], [2], [3], [4], [5], [6].

NoCs have been utilized not only in high-performance microarchitectures, but also in cost-effective embedded devices used mostly in consumer equipment such as set-top boxes or mobile wireless devices. Since such embedded applications often demand very tight design constraints in terms of cost and performance, the silicon budget available for their on-chip network infrastructure should be modest. On the other hand, NoCs are able to exploit the enormous amount of wire resources, unlike interchip interconnects whose bandwidth is usually limited by the pin-count limitation problems outside the chip. If we use a 0.1-μm CMOS technology with a 0.5-μm minimum wire pitch, for example, a 3 mm $\times$ 3 mm tile can exploit up to 6,000 wires on each metal layer, as illustrated in [1]. Finding the on-chip networks that effectively use large numbers of wires to ensure a low-latency and high-throughput communication with a modest silicon budget is thus essential for rapidly evolving embedded devices.

A 2D mesh [4], [5], [6] and torus [1] have been employed as typical on-chip interconnects, because their grid-based regular arrangements are intuitively considered to match the 2D VLSI layout. On the other hand, constant attention has been focused on tree-based topologies, because of their relatively short hop count, which enables a lower latency communication than a mesh and torus. The performances and costs of tree-based networks, as well as grid-based ones, have been widely studied [7], [8], [9], and a tree-based network achieves at least as high a throughput as a mesh one for equivalent chip sizes. In addition, since various on-chip interconnects use a tree-based structure [10], [11], [12], we mainly focused on trees for cost-efficient on-chip networks.

In this paper, we propose a novel tree-based interconnection network called Fat H-Tree that is unlike the existing interconnects mentioned above and is an attractive alternative to the other tree-based topologies. A Fat H-Tree has a torus structure, which is formed by combining only two H-Trees, and it achieves just as high a performance as the torus by using a smaller network logic.

We present three deadlock-free routing schemes of Fat H-Tree for different purposes. One of them can be used to partition a Fat H-Tree into two separated H-Tree networks, each of which can be used for different purposes. It is useful to transfer different types of traffic (e.g., data and control packets) separately, as in some recent on-chip systems [4], [5]. The other two routing schemes combine the two networks of a Fat H-Tree in order to obtain the better hop count and throughput.

In addition, the Fat H-Tree can be used as the network architecture for 3D ICs that stack multiple wafers or dice by using vertical interconnects [13], [14], [15]. We propose a 3D

- *H. Matsutani and H. Amano are with the Department of Information and Computer Science, Faculty of Science and Technology, Keio University, 3-14-1 Hiyoshi, Kouhoku-ku, Yokohama 223-8522, Japan. E-mail: {matutani, hunga}@am.ics.keio.ac.jp.*
- *Y. Yamada is with Toshiba Corporation, Komukai-Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan. E-mail: yutaka@isl.rdc.toshiba.co.jp.*
- *M. Koibuchi is with the Infrastructure Systems Research Division, National Institute of Informatics (NII), 2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan. E-mail: koibuchi@nii.ac.jp.*
- *D.F. Hsu is with the Department of Computer and Information Sciences, Fordham University, 113 West 60th Street, New York, NY 10023. E-mail: hsu@cis.fordham.edu.*

Manuscript received 27 June 2008; revised 4 Oct. 2008; accepted 8 Oct. 2008; published online 20 Oct. 2008.
Recommended for acceptance by B. Parhami.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-06-0246.
Digital Object Identifier no. 10.1109/TPDS.2008.233.

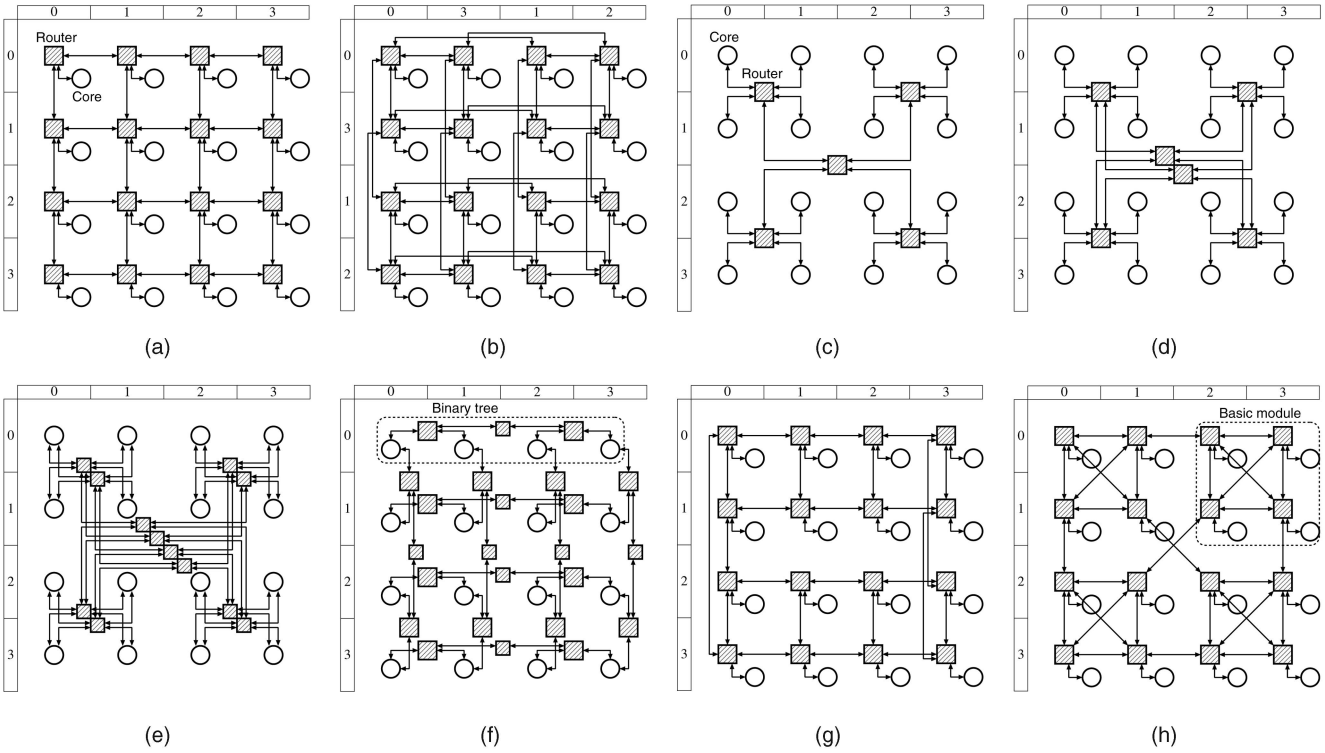

Fig. 1. Two-dimensional layouts of typical interconnects (16-core). (a) Two-dimensional mesh. (b) Two-dimensional torus. (c) H-Tree/Fat Tree (1, 4, 1). (d) Fat Tree (2, 4, 1). (e) Fat Tree (2, 4, 2). (f) MoT. (g) Spidergon. (h) WK recursive (4, 2).

layout scheme of Fat H-Tree that integrates both intrawafer and interwafer networks, in order to reduce its wire length, wire delay, and energy consumed for driving wires.

Finally, this paper shows the following detailed evaluations of Fat H-Tree: 1) the performance of Fat H-Tree is evaluated using real application traces, and the result is compared with those of other tree-based networks, 2) the network logic area and wire resources for Fat H-Tree are estimated based on a typical implementation of NoC routers using a 90-nm standard cell library, and 3) the energy consumption is estimated based on the gate-level power analysis of the routers and wires.

In this paper, Section 2 surveys on-chip network topologies. Sections 3 and 4 propose Fat H-Tree and its 2D layout. Section 5 proposes deadlock-free routing schemes, and Section 6 introduces a 3D layout scheme of Fat H-Tree. Section 7 discusses the cost and performance advantages of Fat H-Tree over typical tree-based networks, based on their topological properties. The theory is demonstrated through simulations in Section 8. Section 9 concludes this paper.

2 Related Work

Fig. 1 shows typical on-chip networks, where a white circle represents a processing core and a shaded square represents a router connecting other routers or cores.

The simplest tree-based topology is the H-Tree in which each router (except for the top-rank router) has one upward and four downward connections (Fig. 1c). However, the links and routers around the root of the tree are frequently congested due to its poor bisection bandwidth.

To mitigate the congestion around the root of the tree, a Fat Tree enhances the number of connections toward the root [7]. As stylized in [7] and [9], various forms of Fat Tree can be created, and they can be expressed with a triple (p, q, c), where p is the number of upward connections, q is the number of downward connections, and c is the number of upward connections that each core has. Fig. 1e shows a typical Fat Tree (2, 4, 2), in which each router (except for top-rank routers) has two upward and four downward connections and each core has two upward connections. This is the network architecture used in CM-5 [16]. Fig. 1d shows a smaller version labeled as (2, 4, 1), which means that every core has only one upward connection. Note that a Fat Tree (1, 4, 1) is identical to the H-Tree.

With a large p and q, the total bandwidth of Fat Tree is enlarged. However, the degree of a router becomes $p + q$, and so, the hardware requirements for the router and its wires are also increased. Using a large c can improve the bisection bandwidth of Fat Tree almost linearly. That is, the bisection bandwidth can be doubled when c is doubled. However, the number of routers is almost doubled, and it requires a lot of routers and wires between them. Considering the number and degree of routers required for Fat H-Tree, we selected Fat Trees (1, 4, 1), (2, 4, 1), and (2, 4, 2) in this paper.

A Fat H-Tree has a torus structure, which is formed by combining only two H-Trees, and it offers performance comparable to the torus by using a smaller network logic. Interconnection networks that have both tree and grid structures have been researched for large-scale parallel machines. The Recursive Diagonal Torus (RDT) [17] is an extended hierarchical torus that also has the tree property. However, since RDT was originally designed for massively parallel machines, its node degree is high (e.g., at least eight), so its connection structure tends to be costly in a microarchitecture domain, and its layout on a chip is also difficult.

A Mesh-of-Trees (MoT) network [18], [9] also has properties of both a mesh and a tree. In an MoT network, starting

with a mesh of processing cores, the cores in each row and column of the mesh are connected by a tree, as shown in Fig. 1f. Although it can utilize the traffic locality, it requires a larger number of routers than other topologies in Fig. 1 do.

The butterfly network (k-ary n-fly) [19] can be efficiently mapped onto a 2D VLSI by utilizing high-radix routers [20]. In the flattened butterfly [21], routers in each row of a conventional butterfly are combined into a single router. Although notable improvements such as adding bypass channels for nonminimal routing with minimal increase in power have been proposed for it [21], it is topologically similar to the conventional butterfly.

The Spidergon topology, which is a bidirectional ring with additional channels that connect diagonal counterparts in the ring, has been proposed for cost-effective on-chip networks [22]. It can be efficiently mapped onto a chip, as shown in Fig. 1g. Although it has a good cost-performance property, its average hop count considerably increases as the number of nodes increases, even though it provides diagonal links to mitigate the increase of diameter compared to a conventional ring.

The 2D shuffle-exchange mesh (SEM) [23] is formed by applying the conventional shuffle-exchange structure in each row and each column of a network. Compared with a traditional mesh, it can reduce the diameter with the same cost by replacing some links with nonadjacent connections based on the shuffle and exchange operations. The optimal 3D layout scheme of 2D SEM that can reduce nearly 30 percent of the link power has also been proposed [24].

In addition, the $\mathrm{WK}(d,t)$ network [25] (Fig. 1h) has been researched for on-chip purposes [26]. It can be constructed by grouping basic modules, each of which is a d-node complete graph, recursively for t times. Thorough comparisons between a WK recursive and a mesh are provided in [26]. Although a WK recursive provides lower latency than a same-sized mesh for low traffic loads, its saturated throughput is comparable or inferior to the mesh.

A large number of network topologies have been proposed so far, but those employed in practical systems are limited to some well-known topologies such as a 2D mesh, a torus, and Fat Trees. These well-known topologies have been used as benchmarks for newly invented ones; therefore, the Fat H-Tree topology proposed here is also compared with a 2D mesh, a torus, and Fat Trees in this paper.

In our previous work [27], [28], [29], the concept of Fat H-Tree was defined, and some results have been obtained. In [27], the idea of Fat H-Tree was originally proposed as an interconnection architecture of dynamically reconfigurable processor arrays. In [28], it was evaluated in terms of cost, performance, and dynamic energy. Its 3D layout scheme was proposed in [29]. This paper is the extended version of the previous work, which revises the definition, routing algorithms, and layout schemes of Fat H-Tree. It also shows new evaluation results in terms of the application throughputs and energy consumption with an advanced CMOS technology.

3 Definition of Fat H-Tree

A Fat H-Tree is a novel tree-based interconnection network with a torus structure, which is formed by combining two H-Tree networks, called **red tree** and **black tree**. Similar to the Fat Tree (2, 4, 2) in Fig. 1e, every processing core in a Fat H-Tree has two ports: one for connecting to the red tree and the other one for the black tree. The network interface (NI) in the Fat H-Tree has a special function to forward packets from red to black and vice versa. This property provides torus-like alternative paths in the Fat H-Tree, which greatly improves its performance (see Section 8.5).

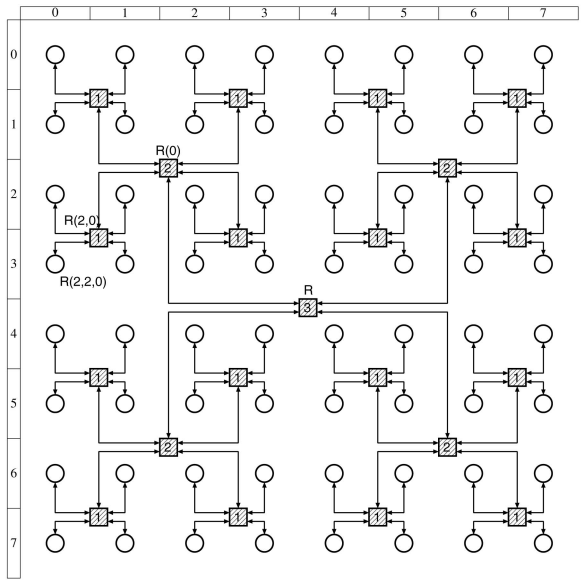

Fig. 2. Red tree.

Assume that 4^n cores are aligned in a $2^n \times 2^n$ 2D grid square, and 2D coordinates (x_{2D}, y_{2D}) are assigned to each core. We call such a core a rank-0 router from the network point of view.

Definition 1 (red tree). *For a rank-0 router (x_{2D}, y_{2D}), the red-tree coordinates $R(r_0, r_1, \ldots r_{n-1})$ are assigned as follows:*

$$r_i = \left(\lfloor x_{2D}/2^i \rfloor \bmod 2\right) + 2 \times \left(\lfloor y_{2D}/2^i \rfloor \bmod 2\right). \quad (1)$$

For each i from 0 to $n-1$, four rank-i red routers $R(r_i, \ldots r_{n-1})$ that have the same part of coordinates $R(r_{i+1}, \ldots r_{n-1})$ are connected to the rank-$(i+1)$ red router labeled $R(r_{i+1}, \ldots r_{n-1})$. The top-rank router in the red tree thus has the coordinates R.

Fig. 2 shows the red tree, where the number in a router (e.g., 1, 2, or 3) represents its rank. For example, red-tree coordinates R, $R(0)$, $R(2,0)$, and $R(2,2,0)$ are shown in the figure.

Definition 2 (black tree). *For a rank-0 router (x_{2D}, y_{2D}), the black-tree coordinates $B(b_0, b_1, \ldots b_{n-1})$ are assigned as follows:*

$$b_i = \left(\lfloor (X-1)/2^i \rfloor \bmod 2\right) + 2 \times \left(\lfloor (Y-1)/2^i \rfloor \bmod 2\right), \quad (2)$$

where $X = 2^n$ if $x_{2D} = 0$ and $X = x_{2D}$ otherwise, and $Y = 2^n$ if $y_{2D} = 0$ and $Y = y_{2D}$ otherwise. For each i from 0 to $n-1$, four rank-i black routers $B(b_i, \ldots b_{n-1})$ that have the same part of coordinates $B(b_{i+1}, \ldots b_{n-1})$ are connected to the rank-$(i+1)$ black router labeled $B(b_{i+1}, \ldots b_{n-1})$. The top-rank router in the black tree thus has the coordinates B.

Fig. 3 shows the black tree, which is located at the lower right of the red tree. For example, black-tree coordinates B, $B(2)$, $B(1,2)$, and $B(0,1,2)$ are shown in the figure.

Definition 3 (Fat H-Tree). *On $2^n \times 2^n$ rank-0 routers, a Fat H-Tree is formed by an n-rank red tree and an n-rank black tree.*

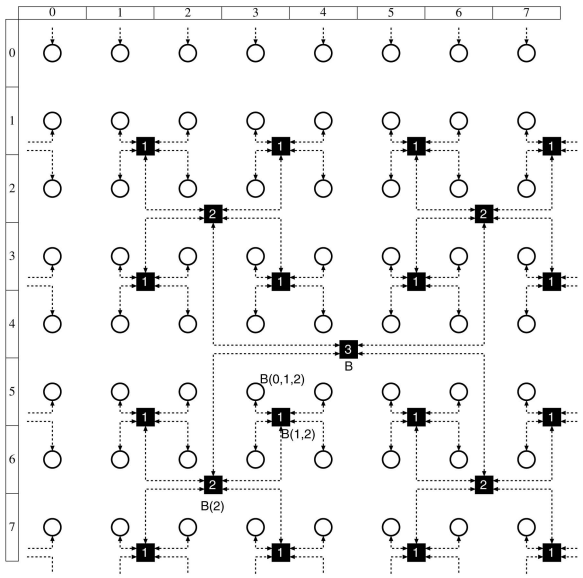

Fig. 3. Black tree.

As shown in Fig. 4, the Fat H-Tree has a torus structure, which is formed with rank-0 and rank-1 routers in both trees. Note that the rank-2 or upper routers in the Fat H-Tree are omitted in the figure, for ease of understanding.

4 Two-Dimensional Layout Scheme

We propose a 2D layout scheme based on a folding technique for Fat H-Tree to avoid long feedback links laid across the chip, such as links between top routers and bottom routers or rightmost routers and leftmost routers, as shown in Fig. 6. To shorten these long links, a Fat H-Tree can be folded in the same manner as folded rings or tori.

Assume that the number of cores in a given Fat H-Tree is $2^n \times 2^n$ and the 2D coordinates (x_{2D}, y_{2D}) are assigned to each core. For the folded layout, the 2D coordinates of each core are transformed into the folded 2D coordinates (x_{2Dfold}, y_{2Dfold}) as follows:

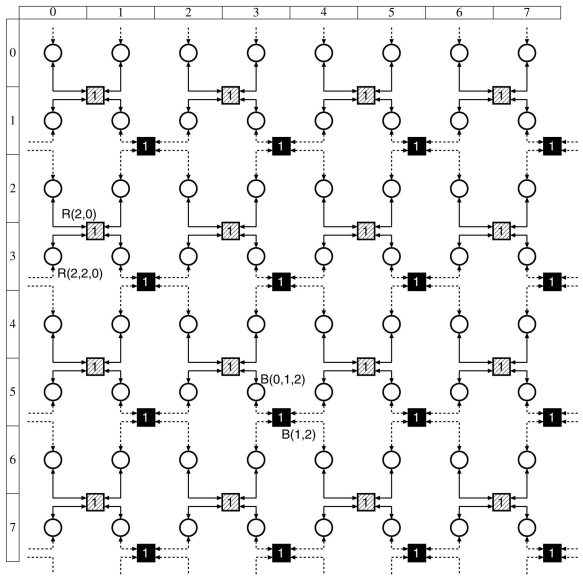

Fig. 4. Fat H-Tree (rank-2 or upper routers are not shown).

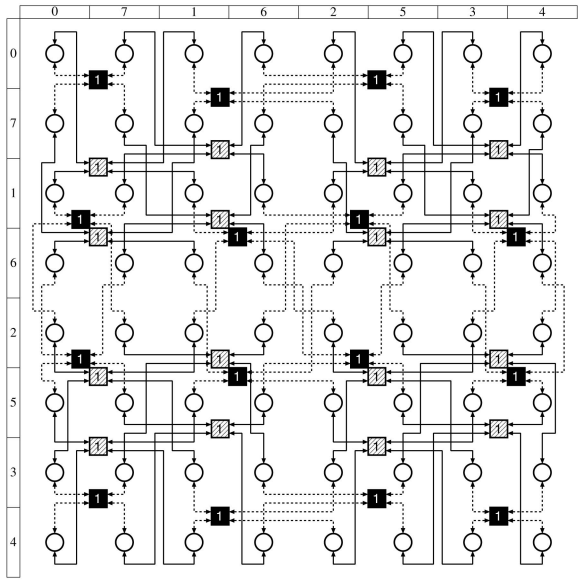

Fig. 5. Folded Fat H-Tree (rank-2 or upper routers are not shown).

$$x_{2Dfold} = \begin{cases} 2x_{2D}, & x_{2D} < 2^{n-1}, \\ -2x_{2D} + 2^{n+1} - 1, & x_{2D} \geq 2^{n-1}, \end{cases}$$
$$y_{2Dfold} = \begin{cases} 2y_{2D}, & y_{2D} < 2^{n-1}, \\ -2y_{2D} + 2^{n+1} - 1, & y_{2D} \geq 2^{n-1}. \end{cases} \quad (3)$$

As shown in Fig. 7, the order of the cores is changed so that every link is connected to the next neighboring cores (except for edge cores). This folding technique is applied to both red and black trees. Then, we can obtain a folded layout of a Fat H-Tree, in which a series of cores in both x and y directions are interleaved (Fig. 5). Obviously, Figs. 4 and 5 are topologically equivalent. The total wire length of the folded layout is analyzed in Section 7.4.

5 Deadlock-Free Routing Schemes

Packet routing is a crucial factor for making the best use of the network resources on a Fat H-Tree. We propose three

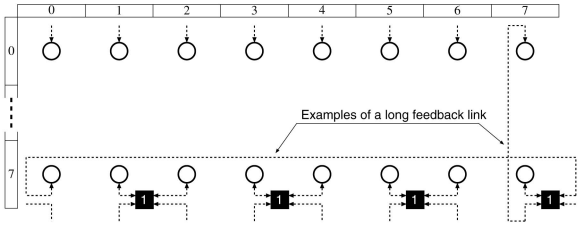

Fig. 6. A part of black tree without folding.

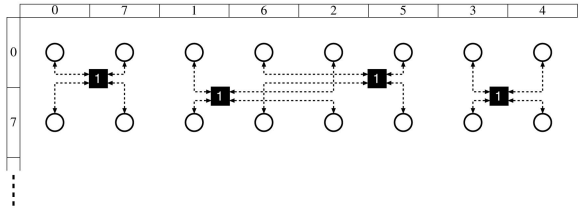

Fig. 7. A part of black tree with folding.

deadlock-free routing schemes that are based on the following rule.

Definition 4 (Fat H-Tree routing). *All routing schemes of Fat H-Tree employed in this paper use virtual channels under the following rule: starting from virtual-channel number zero, the current virtual-channel number is incremented only when a packet is forwarded from red to black via a rank-0 intermediate router.*

Based on this rule, the following three routing schemes are designed for different purposes: network partitioning, exploiting minimal paths, and torus-based paths.

Single-Tree Routing (STR). The source node selects either the red or black tree when it sends a packet, and the routing is done with the selected single tree. Like the common up*/down* routing, packet transfers from downward direction (moving toward a leaf) to upward direction (moving toward a root) are prohibited in the tree.

Minimal Routing (MIN). MIN does not prohibit any packet transfers between trees. It thus can exploit the shortest paths for all source-destination pairs, which are found by the Dijkstra's algorithm. This routing scheme requires $\lfloor H_{max}/4 \rfloor + 1$ virtual channels to guarantee the deadlock freedom, because the longest path changes the trees from red to black up to $\lfloor H_{max}/4 \rfloor$ times, where H_{max} is the maximum hop count of this scheme.

Torus Routing (TOR). TOR uses only the torus structure formed with rank-0 and rank-1 routers, as illustrated in Fig. 4. Therefore, all packet transfers between rank-1 and rank-2 routers are prohibited. This routing scheme requires $\lfloor H_{max}/4 \rfloor + 1$ virtual channels, where H_{max} is the maximum hop count of this scheme.

Although STR suffers from nonminimal paths, it can physically partition the network resources into two subnetworks, both of which provide connectivity between all pairs of processing cores, for different purposes (e.g., prescheduled and dynamic networks in TILE64 [4]). It can also be used for fault tolerance [30]. MIN is a minimal routing, which usually offers high-throughput and low-latency communications. TOR is completely free from the tree's weak point (i.e., root bottleneck), but its average hop count will increase, because it does not use all the network resources. We can further reduce the chip footprint if we use only TOR, since it uses only rank-0 and rank-1 routers.

Theorem. *Any sets of routing paths that are based on STR, MIN, and TOR are deadlock free.*

Proof. Any sets of routing paths that are based on STR, MIN, and TOR guarantee deadlock freedom, because no cyclic dependency occurs as follows:

1. No cyclic dependency is formed within each tree.
2. No cyclic dependency is formed across trees, because packets are passed from the red tree to the black tree by virtual-channel transition in increasing order. □

In a 16-core Fat H-Tree, H_{max} is four when MIN or TOR is used; thus, two virtual channels are required for these schemes. In a 64-core Fat H-Tree, the H_{max} of MIN is six and that of TOR is eight; therefore, MIN requires two virtual channels while TOR needs three. Comparing performance of routing algorithms with different numbers of virtual channels is unfair, since each router requires different amounts of hardware.

Fortunately, although the complete TOR in a 64-core Fat H-Tree needs three virtual channels, torus-based routing paths between most source-destination pairs require only two virtual channels. Thus, all torus-based paths that require three virtual channels are replaced with other routing paths that use only two or less virtual channels between the same source-destination pairs. Such a hybrid scheme with two virtual channels is used in the performance evaluation instead of the complete TOR in order to fairly compare its performance with other routing schemes that use two virtual channels (see Section 8.5).

The path calculation of each routing scheme can be completed with the following procedure: 1) impose the prohibited turns according to the selected routing on a directed graph corresponding to a Fat H-Tree, and 2) search for the shortest paths between a given source-destination pair by using Dijkstra's algorithm so as not to use the prohibited turns defined by the selected routing scheme.

STR and TOR can be implemented with simple combinational logic that can calculate their route in a single cycle, while MIN is too complex to be used as an online algorithm. In this paper, these routing schemes are implemented as source routing, which is one of popular routing implementations used in NoCs [1], [6].

In this section, three routing schemes were proposed for Fat H-Tree. These routing schemes should be carefully selected in order to meet the requirements of the target application. By preparing multiple routing paths for a single source-destination pair, it is also possible to dynamically and partially update the routing scheme. The throughput results of these routing schemes, which can be used for the routing selection, are shown in Section 8.5.

6 Three-Dimensional Layout Scheme

The Fat H-Tree can be used as the network architecture for 3D ICs that stack multiple tiers (a tier refers a wafer or a die in a 3D IC) using vertical interconnects [13], [14], [15]. In this section, we propose a 3D layout scheme of trees that divides the original planar network into several tiers and connects them by using vertical interconnects in order to reduce their wire length, wire delay, and the energy consumed for driving wires.

We assume to use through-wafer via technology, which is expected to offer both a very high density in the vertical interconnects and a very short distance between the wafers. The distance between wafers can range from 5 μm to 50 μm [31], which is much shorter than the wire length between the cores on a tier, and the pitches of a through-wafer via can range from a 1-μm to a 10-μm square [13], [14], [31], depending on the manufacturing process such as the wafer-to-wafer alignment. By using the 3D layout scheme, the longest links in the original 2D layout of trees can be replaced with vertical links whose lengths are very short (e.g., 5 μm to 50 μm).

A 3D layout scheme of Fat Trees is preliminarily introduced in Section 6.1, and then, that of Fat H-Tree is proposed in Section 6.2.

6.1 Fat Trees

Four-split scheme. The following is the procedure that splits a given Fat Tree into four tiers:

1. *Chip partitioning.* Assume that the number of cores in a given Fat Tree is $2^n \times 2^n$ and the 2D coordinates (x_{2D}, y_{2D}) are assigned to each core. For a 3D layout,

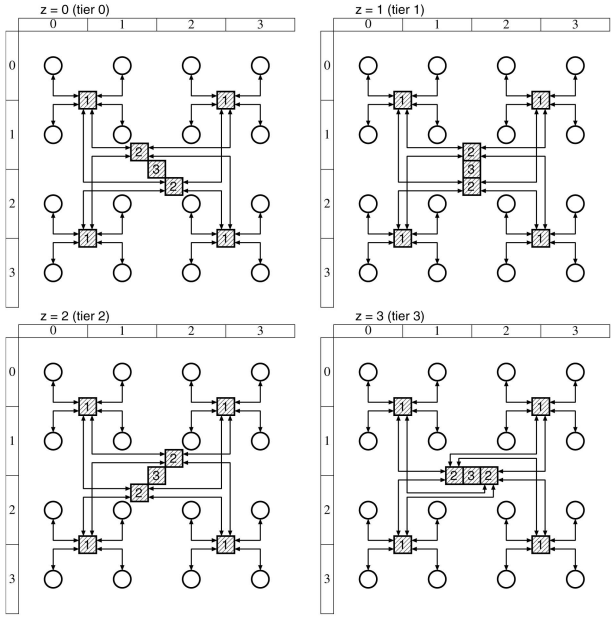

Fig. 8. Splitting a 64-core Fat Tree (2, 4, 1) into four.

the 2D coordinates of each core are transformed into the 3D coordinates (x_{3D}, y_{3D}, z_{3D}) as follows:

$$\begin{aligned} x_{3D} &= x_{2D} \bmod 2^{n-1}, \\ y_{3D} &= y_{2D} \bmod 2^{n-1}, \\ z_{3D} &= 2 \times \lfloor y_{2D}/2^{n-1} \rfloor + \lfloor x_{2D}/2^{n-1} \rfloor. \end{aligned} \tag{4}$$

For example, a 64-core Fat Tree (2, 4, 1) can be divided into four tiers, each of which has 16 cores, as shown in Fig. 8.

2. *Allocation of routers*. The routers are evenly distributed across all tiers so that each tier has the same number of routers for each tree level. In Fig. 8, each tier has four rank-1 routers, two rank-2 routers, and a single rank-3 router.
3. *Allocation of vertical links*. The routers are connected by horizontal wires and/or vertical links so as to minimize the wire length. In order to connect the routers on different tiers, vertical links such as through-wafer vias [14] are placed between the tiers.
4. *Reallocation of routers*. The locations of the routers are adjusted so as not to overlap the vertical links, since vertical links consume the overhead area, which is much smaller than the router area but cannot overlap each other. In Fig. 8, each rank-3 router is placed in the center of its corresponding tier, and rank-2 routers are placed closely around the rank-3 router.

Steps 3 and 4 are performed so that the total wire length or the longest wire length in a given network is minimized as long as the vertical interconnects do not overlap each other. They can be done by hand or optimized by some placement algorithms such as the simulated annealing.

As a consequence, every top-rank link, which requires long wires to connect rank-2 and rank-3 routers in the original 2D layout, is replaced by a very short vertical link, resulting in considerable wire length reduction. This layout scheme can be used in other types of Fat Trees such as (2, 4, 2). We call this scheme a "four-split scheme," since the original 2D layout is divided into four pieces.

Two-split scheme. More generally, a given Fat Tree can be divided into 2^i tiers, by combining the four-split scheme and the two-split scheme, where i is a positive integer. When a given Fat Tree is divided into two pieces (i.e., two-split), the 2D coordinates of each core are transformed as follows:

$$\begin{aligned} x_{3D} &= x_{2D}, \\ y_{3D} &= y_{2D} \bmod 2^{n-1}, \\ z_{3D} &= \lfloor y_{2D}/2^{n-1} \rfloor. \end{aligned} \tag{5}$$

The other steps (i.e., the allocations of routers and vertical links) are the same as those in the four-split scheme. Compared with the original 2D layout, the longest link length in the 3D layout folded with the two-split scheme is reduced to approximately $\sqrt{1/2}$.

6.2 Fat H-Tree

By extending the 3D layout scheme of Fat Trees mentioned above, we propose a 3D layout scheme of Fat H-Tree that can keep a torus structure intact even though the Fat H-Tree is partitioned into several tiers being connected by vertical links.

Four-split scheme. Here is the procedure for splitting a given Fat H-Tree into four tiers:

1. *Chip partitioning*. Assume that the number of cores in a given Fat H-Tree is $2^n \times 2^n$ and the 2D coordinates (x_{2D}, y_{2D}) are assigned to each core. For the 3D layout, the 2D coordinates of each core are transformed into the 3-D coordinates (x_{3D}, y_{3D}, z_{3D}) as follows:

$$\begin{aligned} x_{3D} &= \begin{cases} x_{2D} \bmod 2^{n-1}, & x_{2D} < 2^{n-1}, \\ -(x_{2D} \bmod 2^{n-1}) + 2^{n-1} - 1, & x_{2D} \geq 2^{n-1}, \end{cases} \\ y_{3D} &= \begin{cases} y_{2D} \bmod 2^{n-1}, & y_{2D} < 2^{n-1}, \\ -(y_{2D} \bmod 2^{n-1}) + 2^{n-1} - 1, & y_{2D} \geq 2^{n-1}, \end{cases} \\ z_{3D} &= 2 \times \lfloor y_{2D}/2^{n-1} \rfloor + \lfloor x_{2D}/2^{n-1} \rfloor. \end{aligned} \tag{6}$$

The original 2D layout of Fat H-Tree is folded in both the vertical and horizontal directions, resulting in four folded pieces. For example, the 2D layout of a 64-core red tree (Fig. 2) is mapped onto a 3D space consisting of four tiers, as shown in Fig. 9. Similarly, the 2D layout of a 64-core black tree is transformed into the 3D layout shown in Fig. 10.

2. *Allocation of routers*. The routers are evenly distributed across all tiers. In this example, each tier has eight rank-1 routers (four for the red tree and the others for the black) and two rank-2 routers (one for the red tree and the other for the black). In addition, tier 1 has a rank-3 red-tree router, whereas tier 2 has a rank-3 black one.
3. *Allocation of vertical links*. The vertical links such as the through-wafer vias are placed so as to shorten the horizontal wire length between two linked routers mounted on different tiers. Notice that several links in the black tree (Fig. 10) are labeled "to tier n." This means that such a link is connected to an associated rank-1 black-tree router mounted on tier n.

Although we have considered the red and black trees separately, the 3D layout of Fat H-Tree is formed by superimposing the 3D layouts of the red and black trees on the same tiers.

In order to show that the 3D layout of Fat H-Tree can maintain its torus (ring) structure intact, we must illustrate that a packet that moves in the $y+$ direction from core $(1, 0, 0)$ goes around the ring, and then, it can reach core $(1, 0, 0)$ again.

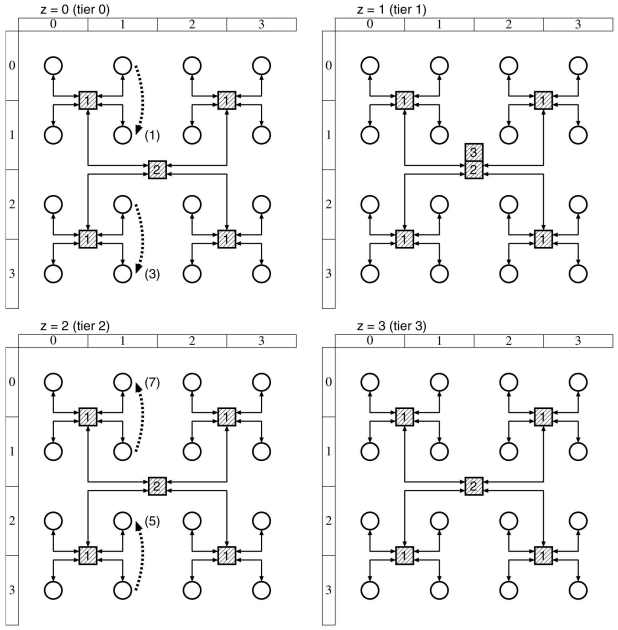

Fig. 9. Splitting a 64-core red tree into four.

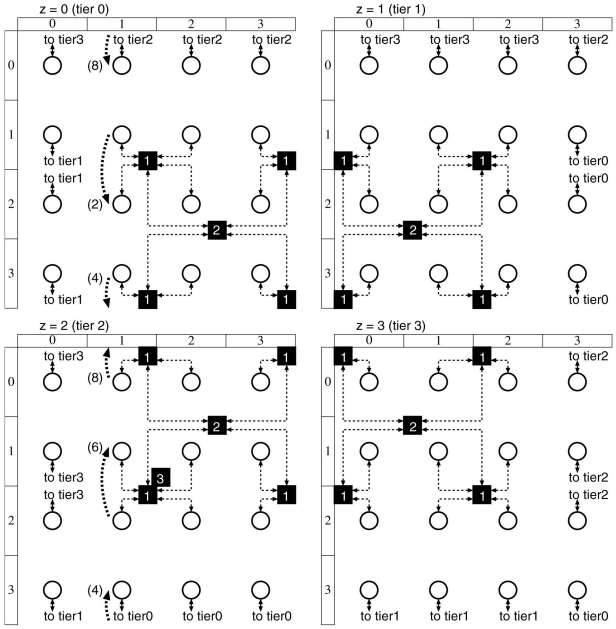

Fig. 10. Splitting a 64-core black tree into four.

After moving two hops in the $y+$ direction from core (1, 0, 0), the packet reaches core (1, 1, 0), as shown by arrow 1 in Fig. 9. The next core from core (1, 1, 0) is core (1, 2, 0), as shown by arrow 2 (Fig. 10), and the next is core (1, 3, 0), as shown by arrow 3 (Fig. 9). Then, the packet moves up to core (1, 3, 2) on tier 2, as shown by arrow 4. On tier 2, in the same way, the packet goes through core (1, 3, 2), core (1, 2, 2), core (1, 1, 2), and core (1, 0, 2). Finally, the packet gets back to core (1, 0, 0) on tier 0, as shown by arrow 8.

As mentioned above, the 3D layout of a given Fat H-Tree is obtained by folding the original Fat H-Tree one or more times until the number of folded pieces meets the number of tiers the 3D chip has (e.g., two times for four-split).

Two-split scheme. Just as for Fat Trees, a given Fat H-Tree can be divided into 2^i tiers, by combining the four-split and two-split schemes of Fat H-Tree. In the case of the two-split scheme, the 2D coordinates are transformed as follows:

$$\begin{aligned} x_{3D} &= x_{2Dfold}, \\ y_{3D} &= \begin{cases} y_{2D} \bmod 2^{n-1}, & y_{2D} < 2^{n-1}, \\ -(y_{2D} \bmod 2^{n-1}) + 2^{n-1} - 1, & y_{2D} \geq 2^{n-1}, \end{cases} \\ z_{3D} &= \lfloor y_{2D} / 2^{n-1} \rfloor. \end{aligned} \tag{7}$$

Note that every ring structure in the y direction is formed across two tiers, whereas that in the x direction is formed within a single tier. In the two-split scheme, therefore, every ring in the x direction must be folded within a single tier, just as in the 2D layout of rings or tori. That is, a series of cores in the y direction are placed adjacently, whereas those in the x direction are interleaved.

The other steps are the same as those for the Fat H-Tree's four-split scheme.

7 Topological Properties

In this section, we discuss the cost and performance advantages of Fat H-Tree over typical tree-based networks, based on their topological properties in terms of the channel bisection, average hop count, number of routers, and total length of links. The properties were confirmed by conducting simulations, as will be described in the next section.

7.1 Ideal Throughput

The ideal throughput of a network is the data acceptance rate that would result from a perfectly balanced routing and a flow control with no idle cycles; it is calculated as [19]

$$\Theta_{ideal} \leq \frac{2bB_c}{N}, \tag{8}$$

where N is the number of cores, b is the channel bandwidth, and B_c is the channel bisection of the network. Table 1 shows the channel bisection of typical networks. Again, the number of cores is $N = 2^n \times 2^n$; therefore, the number of ranks in an N-core tree is $\log_4(N) = \log_4(4^n) = n$.

The channel bisection of an N-core Fat H-Tree is $2^{n+2} + 8$, where the second term 8 corresponds to that in two H-Tree networks (i.e., red and black trees), and the first term 2^{n+2} is due to the torus structure of the Fat H-Tree. Compared with the ideal throughput of simply doubled H-Trees, the ideal throughput of a Fat H-Tree is greatly improved by the torus structure. Section 8.5 shows some case studies on the application throughputs on Fat H-Tree by using a flit-level simulator.

TABLE 1
Channel Bisection B_c

	N-core	16-core	64-core	256-core
H-Tree	4	4	4	4
Fat Tree (2,4,1)	2^{n+1}	8	16	32
Fat Tree (2,4,2)	2^{n+2}	16	32	64
Fat H-Tree	$2^{n+2}+8$	24	40	72
2-D Mesh	2^{n+1}	8	16	32
2-D Torus	2^{n+2}	16	32	64

TABLE 2
Average Hop Count H_{ave}
(Router-Core Hop Is Included as 1 Hop)

	Routing	16-core	64-core	256-core
H-Tree, Fat Tree(2,4,*)	up*/down*	3.60	5.43	7.36
Fat H-Tree	STR	3.20	5.02	7.07
Fat H-Tree	MIN	3.20	4.84	6.88
Fat H-Tree	TOR	3.20	5.65	10.84
2-D Mesh	DOR	4.67	7.33	12.67
2-D Torus	DOR	4.13	6.06	10.03

7.2 Average Hop Count

The number of source-destination pairs in an N-core network is $N^2 - N$; thus, the average hop count in the network is

$$H_{ave} = \frac{1}{N^2 - N} \sum_{x,y \in N} H_{(x,y)}, \tag{9}$$

where $H_{(x,y)}$ is the hop count from core x to core y. Table 2 shows the average hop count of typical networks for uniform random traffic, in which each source sends equally to each destination. The average hop count depends on whether the routing includes nonminimal paths. In the table, STR and TOR in Fat H-Tree are nonminimal. MIN is a minimal routing, and it achieves a 6.5 percent-11.1 percent shorter average hop count compared with Fat Trees.

Note that the average hop count varies depending on the traffic pattern and task mapping. The average hop counts optimized for various applications are shown in Figs. 15 and 16.

7.3 Number of Routers

The number of routers in a chip affects the network logic area and the implementation cost.

Assuming that the number of downward connections q is four, the number of routers in an n-rank H-Tree network, R_{ht}, is

$$R_{ht} = (q^n - 1) \,/\, (q - 1) = (4^n - 1) \,/\, 3. \tag{10}$$

Similarly, the number of routers in a Fat Tree (2, 4, 1) network, R_{ft1}, is

$$R_{ft1} = (q^n - 2^n) \,/\, (q - 2) = (4^n - 2^n) \,/\, 2. \tag{11}$$

A Fat Tree (2, 4, 2) contains twice as many routers as are in the Fat Tree (2, 4, 1); therefore, the number of routers in the Fat Tree (2, 4, 2) is $R_{ft2} = 2R_{ft1}$.

A Fat H-Tree contains two H-Trees, so the number of routers it has is $R_{fht} = 2R_{ht}$.

Table 3 lists the number of routers in typical networks. The number of routers in a Fat H-Tree is less than that in a Fat Tree (2, 4, 2), yet the Fat H-Tree outperforms the Fat Tree in terms of ideal throughput and average hop count, as shown in Sections 7.1 and 7.2. To assess the cost and performance advantages of Fat H-Tree, we need to take into consideration the amount of hardware for each router and NI in addition to the number of routers, since a Fat H-Tree and a Fat Tree (2, 4, 2) require a two-port NI per core, while the others use a one-port NI for each core. In addition, an NI in the Fat H-Tree has a special function that forwards packets from one port to another, and this function would increase the amount of hardware in each NI. In Section 8.1, we present how we implemented an entire Fat-H-Tree-based NoC including NIs, and then, we compare it with other typical networks in terms of the network logic area.

TABLE 3
Number of Routers R

	N-core	16-core	64-core	256-core
H-Tree	$(4^n - 1)/3$	5	21	85
Fat Tree (2,4,1)	$(4^n - 2^n)/2$	6	28	120
Fat Tree (2,4,2)	$4^n - 2^n$	12	56	240
Fat H-Tree	$2(4^n - 1)/3$	10	42	170
2-D Mesh	N	16	64	256
2-D Torus	N	16	64	256

7.4 Total Unit Length of Links

Here, we present the wire length of Fat Trees, Fat H-Tree, and other typical topologies. Assuming that the distance between two neighboring cores aligned in a 2D grid square is 1 unit, we define L as the total unit length of links in a given network. For instance, the 16-core H-Tree network shown in Fig. 1c has 16 1-unit-length links and four 2-unit-length links; thus, its L is 24 units.

In this section, we first show the total unit length of the 2D layout for each topology. Then, we show that of the 3D layout assuming that a given network is divided into four tiers. That is, 16-core, 64-core, and 256-core networks are divided into four tiers, each of which contains 2×2 cores, 4×4 cores, and 8×8 cores, respectively.

Two-dimensional layout. The total unit length of links in an n-rank H-Tree network, $L_{2D,ht}$, can be expressed as

$$L_{2D,ht} = \sum_{i=1}^{n} l_{ht}^i \cdot r_{ht}^i, \tag{12}$$

where l_{ht}^i is the total unit length of links between a rank-i router and its four child routers, and r_{ht}^i is the number of rank-i routers in the H-Tree. Assuming that the number of cores is $N = 2^n \times 2^n$, $l_{ht}^i = 2^{i+1}$ and $r_{ht}^i = N/4^i$, where $1 \le i \le n$. Therefore, (12) can be transformed as follows:

$$L_{2D,ht} = \sum_{i=1}^{n} l_{ht}^i \cdot r_{ht}^i = \sum_{i=1}^{n} 2^{i+1} \cdot \frac{N}{4^i} = 2(N - 2^n). \tag{13}$$

Similarly, the total unit length of links in a Fat Tree (2, 4, 1) network, $L_{2D,ft1}$, is

$$L_{2D,ft1} = \sum_{i=1}^{n} l_{ft1}^i \cdot r_{ft1}^i = \sum_{i=1}^{n} 2^{i+1} \cdot \frac{N}{2^{i+1}} = nN. \tag{14}$$

A Fat Tree (2, 4, 2) has double the number of routers in the Fat Tree (2, 4 ,1), so $L_{2D,ft2} = 2L_{2D,ft1}$.

A Fat H-Tree has two folded H-Tree networks, in which each link, except for the links connecting to the top-rank router, requires twice the wire resources of an ordinary H-Tree. By folding the H-Tree, only the top-rank router and its four child routers can be placed inside a 1 unit $\times$ 1 unit grid square. Thus, the total unit length of links in a Fat H-Tree, $L_{2D,fht}$, can be expressed as follows:

$$l_{fht}^i = \begin{cases} 2l_{ht}^i, & 1 \le i \le n-1, \\ 4, & i = n, \end{cases} \tag{15}$$

TABLE 4
Total Unit Length of Links L for 2D NoCs
(1 Unit = Distance between Neighboring Two Cores)

	N-core	16-core	64-core	256-core
H-Tree	$2(N-2^n)$	24	112	480
Fat Tree (2,4,1)	nN	32	192	1,024
Fat Tree (2,4,2)	$2nN$	64	384	2,048
Fat H-Tree	$8+8(N-2^{n+1})$	72	392	1,800
2-D Mesh	$2(N-2^n)$	24	112	480
2-D Torus	$4(N-2^n)$	48	224	960

$$L_{2D,fht} = \sum_{i=1}^{n} l^i_{fht} \cdot r^i_{fht} = l^n_{fht} \cdot r^n_{fht} + \sum_{i=1}^{n-1} 2^{i+2} \cdot \frac{2N}{4^i} = 8 + 8(N - 2^{n-1}). \quad (16)$$

The total unit lengths of the 2D layouts mentioned above are summarized in Table 4. As for the mesh and torus, we ignore the links between the core and the router, which will increase the total unit length, for the sake of simplicity. Although a Fat H-Tree uses slightly more wire resources compared to the Fat Tree (2, 4, 2) in 16 and 64-core networks, the impact on the chip design is considered modest, because enormous wire resources are available in an NoC, thanks to current CMOS technology, which has eight or more metal layers. In Section 8.2, we discuss the impact of wire demand on Fat H-Tree in a 90-nm CMOS technology.

The longest link in a network affects the wire delay and the number of repeater buffers inserted in the wires. As for the Fat H-Tree, each link, except for the links connecting to the top-rank router, requires twice the wire length of a same-sized H-Tree, while the length of the top-rank link is 1 unit because of its folded layout, as mentioned above. Thus, the longest link length in a Fat H-Tree is the same as those in the H-Tree and the Fat Trees.

Three-dimensional layout. We estimate the total unit length required for the 3D layout of each topology. Since the distance between wafers (i.e., tiers) can range from 5 μm to 50 μm [31], which is much shorter than that of the horizontal links (e.g., 1.0 mm), we do not consider the length of vertical links here for simplicity reasons.

Assuming that an n-rank H-Tree network is divided into four tiers, we first estimate the total link length of its 3D layout, $L_{3D,ht}$. Every top-rank link in its original 2D layout is replaced by a vertical link that connects the routers on different tiers or a very short horizontal link that connects the routers on the same tier. We also do not consider the length of these very short horizontal links. For the 3D layout, the total link length of an n-rank H-Tree can be expressed as follows:

$$L_{3D,ht} = \sum_{i=1}^{n-1} 2^{i+1} \cdot \frac{N}{4^i} = 2(N - 2^{n-1}). \quad (17)$$

For Fat H-Tree, we consider the red and black trees separately. The 3D layout of a red tree is equivalent to that of a same-sized H-Tree; therefore, its total wire length can be expressed with (17). The 3D layout of a black tree includes a same-sized H-Tree, as well as that of a red tree, and it also requires four 2-unit-length links for its top-rank links, as shown in Fig. 10. Therefore, the total wire resources required for the 3D layout of a given Fat H-Tree can be expressed as follows:

TABLE 5
Total Unit Length of Links L for 3D NoCs
(1 Unit = Distance between Neighboring Two Cores)

	N-core	16-core	64-core	256-core
H-Tree	$2(N-2^{n+1})$	16	96	448
Fat Tree (2,4,1)	$(n-1)N$	16	128	768
Fat Tree (2,4,2)	$2(n-1)N$	32	256	1,536
Fat H-Tree	$8+4(N-2^{n+1})$	40	200	904
3-D Mesh	$2(N-2^{n+1})$	16	96	448
3-D Torus	$4(N-2^{n+1})$	32	192	896

$$L_{3D,fht} = L_{3D,ht} + (8 + L_{3D,ht}) = 8 + 4(N - 2^{n+1}). \quad (18)$$

The total unit lengths of the 3D layouts mentioned above are summarized in Table 5. Compared with the original 2D layouts, the 3D layouts reduce their total wire length by 44.5 percent-49.8 percent for Fat H-Tree and by 25.0 percent-50.0 percent for Fat Trees. One of the trees' drawbacks is their wire length, but it can be mitigated by using the 3D layout schemes proposed here. In addition, the 3D layout of Fat H-Tree can reduce more wire resources than that of Fat Tree (2, 4, 2) for large networks. For example, the total unit length of Fat H-Tree is 21.9 percent shorter than that of Fat Tree (2, 4, 2) for 64-core networks in 3D.

Now, we discuss why the 3D layout of Fat H-Tree can reduce more wires than those of Fat Trees. In the case of the 2D layout, a given Fat H-Tree must be folded by interleaving a series of cores in each ring, thus resulting in longer wires between the neighboring cores. In the 3D layout, on the other hand, a given Fat H-Tree does not interleave a series of cores in each ring, since every ring structure can be formed across two tiers without being folded inside a single tier, as shown in Figs. 9 and 10. In other words, the 3D layout of Fat H-Tree, which does not interleave cores, requires fewer wiring resources for connecting neighboring cores compared with the original 2D layout. This is the reason why the 3D layout of Fat H-Tree can reduce more wires compared to Fat Trees.

8 Evaluations

The advantages and disadvantages of Fat H-Tree over Fat Trees are demonstrated through simulations that evaluate their area, wire resources, energy consumption, leakage power, and throughput.

8.1 Hardware Amount

The network logic area in an NoC is mainly composed of routers and NIs that connect a processing core to the network. In this section, the Fat H-Tree is compared with other typical networks in terms of the network logic area.

We implemented a wormhole router that supports various node degrees. We also developed an NoC generator that automatically connects the routers and NIs in the arbitrary network topologies. Using the Synopsys Design Compiler version Y-2006.06-SP2, we synthesized the generated NoC design with ASPLA 90-nm standard cell library and estimated the amount of hardware in the network logic area. The behavior of the synthesized NoC design was confirmed through a gate-level simulation with an operating frequency of 500 MHz.

The router's architecture is fully pipelined. It transfers a header flit through three pipeline stages consisting of the routing computation, virtual-channel and crossbar alloca-

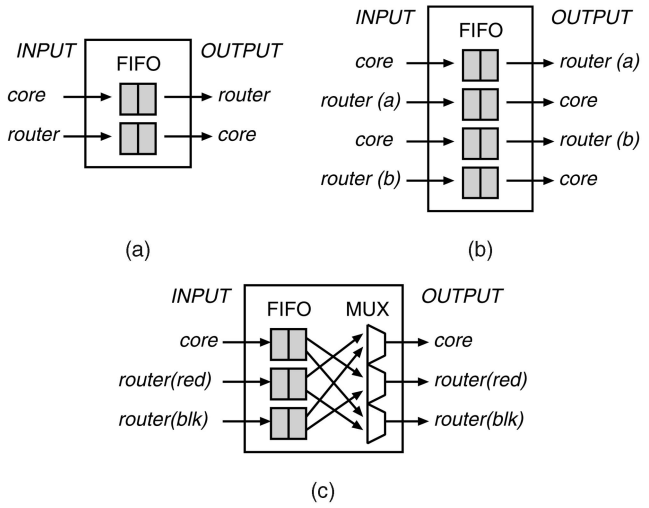

Fig. 11. NI architectures. (a) One-port NI. (b) Two-port NI. (c) Fat H-Tree's NI.

tion, and crossbar traversal. The flit width is set to 64 bits. Each physical channel has two virtual channels, each of which has a FIFO buffer for storing four 64-bit flits. The routing decisions are stored in the header flit prior to packet injection (i.e., source routing); thus, the routing tables that require register files for storing routing paths are not needed in each router, resulting in a low-cost router implementation.

The NI has to be designed to interface between a processing core and the network with a minimum amount of hardware. We implemented a simple NI that employs a two-flit FIFO buffer for both the core-to-network and network-to-core interfaces, as shown in Fig. 11a. Every processing core in a Fat H-Tree and a Fat Tree (2, 4, 2) has two upward connections, while that in a Fat Tree (2, 4, 1) has only one connection. For the Fat Tree (2, 4, 2), we also implemented a two-port NI that has two sets of FIFO buffers (Fig. 11b). In addition, the NI in the Fat H-Tree has to be designed to support a special function to forward packets from one port to another. This function requires a small two-input multiplexer for each of the three output ports in the NI, as shown in Fig. 11c.

Fig. 12 shows the synthesis results of typical 16 and 64-core networks. In the graphs, "HT" represents H-Tree, "FT1" is Fat Tree (2, 4, 1), "FT2" is Fat Tree (2, 4, 2), and "FHT" is Fat H-Tree. Although a Fat H-Tree requires the largest NI area because of its two-port interfaces with the packet forwarding function, the number of routers in a Fat H-Tree is relatively small, as shown in Section 7.3. As a result, a Fat H-Tree is smaller than a 2D mesh, torus, and Fat Tree (2, 4, 2). In particular, a Fat H-Tree consumes 19.8 percent-27.8 percent smaller network logic area than a Fat Tree (2, 4, 2).

8.2 Wire Resources

As shown in Section 7.4, a Fat H-Tree requires slightly more wire resources compared to a Fat Tree (2, 4, 2). However, the impact is considered to be modest, because the enormous wire resources are available in an NoC, thanks to the current process technology that has eight or more metal layers. Here, we first estimate the required wire length for a Fat H-Tree.

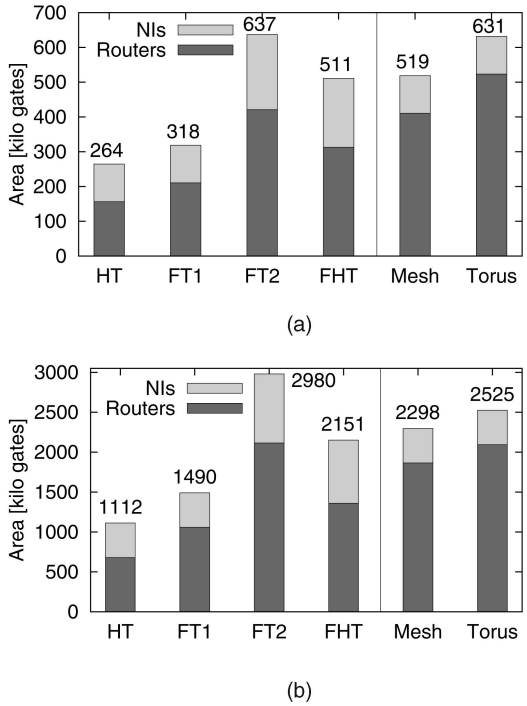

Fig. 12. Network logic area of 16- and 64-core typical networks. (a) 16-core network. (b) 64-core network.

Then, we calculate the utilization ratio of all the wire resources in a chip, in order to demonstrate that the current process technology can provide sufficient wire resources for implementing Fat H-Tree-based NoCs.

The total wire length for a given network can be estimated as

$$W = 2L \times lw, \tag{19}$$

where L is the total unit length of links (bidirectional) in the network, l is the distance between two neighboring cores (e.g., 1.0 mm), and w is the bit width of a physical link.

Two-dimensional layout. Assuming an 8 mm $\times$ 8 mm chip, the distance between two neighboring cores, l, is 2.0 mm for a 16-core network and 1.0 mm for a 64-core network. Based on (19), the required wire lengths for the 16-core and 64-core Fat H-Tree topologies are 18.4 m and 50.2 m, respectively. Those for Fat Tree (2, 4, 2) topologies are 16.4 m and 49.2 m, respectively.

We now shift to the utilization ratio of the wire resources available in a 90-nm CMOS technology. Although current process technologies offer eight or more metal layers, we assume that the on-chip network infrastructure can use only two metal layers, while the application logic (i.e., processing cores) can exploit all layers. Assuming that the minimum wire pitch is 0.5 μm for this technology, the 8 mm $\times$ 8 mm chip has up to 16,000 wires on each metal layer crossing each edge of the chip; thus, up to 256 m of wires would be available in two metal layers in the chip.[1]

A Fat H-Tree utilizes 7.2 percent of the wires in a 16-core network, and it uses only 19.6 percent of wires in a large 64-core network. This also shows that the differences

1. Although this assumption often overestimates the available resources in a chip because certain tracks are used for power and ground [32], it enables us to simply approximate the wire utilization.

between a Fat H-Tree and a Fat Tree (2, 4, 2) are very small (i.e., 0.4 percent-0.8 percent). In addition, the 90-nm process used in this section is not a state-of-the-art technology. The latest technologies such as the 65-nm or 45-nm processes can provide many more wires, which would further ease the wiring issue of Fat H-Tree. Thus, it is clear from the above discussion that the current process technology can provide sufficient wire resources for implementing Fat H-Tree-based on-chip networks.

Three-dimensional layout. The 3D layout of Fat H-Tree reduces the total wire length by 44.5 percent-49.8 percent compared to the original 2D layout, as shown in Table 5. Therefore, the demand of wire resources for Fat H-Tree is mitigated as well.

8.3 Flit Transmission Energy

The Fat H-Tree and other topologies are compared in terms of the average energy consumption needed to transmit a single flit from source to destination (denoted as E_{flit}).[2] It can be estimated as [35]

$$E_{flit} = wH_{ave}(E_{sw} + E_{link}), \tag{20}$$

where w is the flit width, H_{ave} is the average hop count, E_{sw} is the average energy to switch the 1-bit data inside a router, and E_{link} is the 1-bit energy consumed in a link.

We used the Synopsys Power Compiler to extract the E_{sw} of the routers and NIs, which were synthesized with the 90-nm standard cell library (the architectures of the routers and NIs were described in Section 8.1). In the same manner as reported in [33], a fixed workload was applied to the routers and NIs, and their switching activities were captured through gate-level simulations operating at 500 MHz with a 1.0-V core voltage. The gate-level power analysis based on these switching activities shows that E_{sw} is 0.183 pJ for the router, 0.092 pJ for the original NIs (Figs. 11a and 11b), and 0.140pJ for the Fat H-Tree's NI (Fig. 11c).

E_{link} can be calculated as

$$E_{link} = dV^2C_{wire}/2, \tag{21}$$

where d is the average 1-hop distance (in millimeters), V is the supply voltage, and C_{wire} is the wire capacitance per millimeter. C_{wire} can be estimated using the method proposed in [36] and is 300 fF/mm for the semiglobal interconnects in the 90-nm CMOS technology. For instance, E_{link} is 0.150 pJ when the 1-hop distance is 1 mm on the average.

We used 64-bit 16- and 64-core networks placed on an 8 mm $\times$ 8 mm chip. In a 16-core network, H_{ave} tends to be short, but d gets longer, while they are opposite for a 64-core network. We estimated the average 1-hop distance, d, using a flit-level network simulator, as in [28]. Then, we derived E_{flit} based on (20) with the various parameters mentioned above.

Fig. 13 shows the average flit transmission energy, E_{flit}, for 16 and 64-core networks. The evaluation results of the 2D and 3D layouts are shown in a different color in these graphs.

2. E_{flit} is useful to discuss the energy efficiency of various networks, since it is independent of the frequency, device area (standby power), and throughput. On the other hand, the power consumption of routers and NoCs (in Watts) is analyzed in [33] and [34].

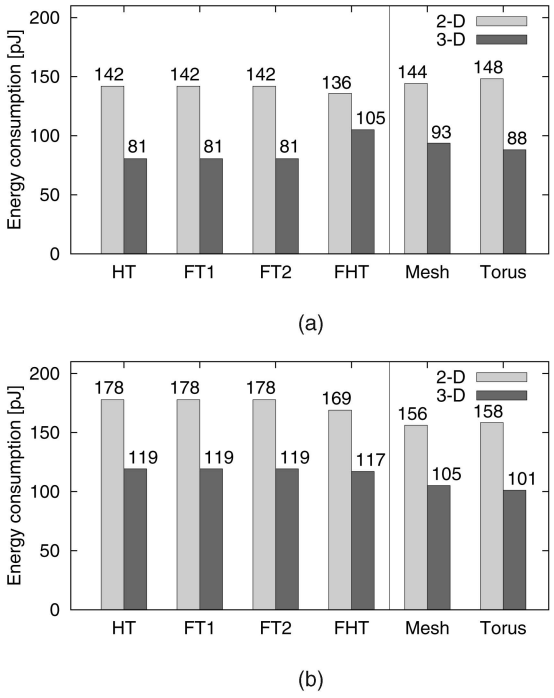

Fig. 13. Average energy consumption to transmit a single flit (flit width = 64 bits). (a) 16-core network. (b) 64-core network.

8.3.1 Two-Dimensional Layout

We first present the results from the 2D layouts. Although the average 1-hop distance, d, of Fat H-Tree is longer than the other tree-based topologies because of its folded layout, its average hop count is the shortest among them (see Section 7.3). As a result, a Fat H-Tree consumes 4.3 percent-5.0 percent less energy than Fat Trees.

8.3.2 Three-Dimensional Layout

We shift to E_{flit} on the 3D layouts of Fat Trees and Fat H-Tree. They are placed on four 4 mm $\times$ 4 mm wafers for this experiment. The capacitance of a vertical link is very small (e.g., 4.34 fF as reported in [14]) compared with that of horizontal links (e.g., 300 fF/mm in the 90-nm technology), so we do not assess the capacitance of the vertical links for simplicity reasons. Fig. 13 also shows the results from the 3D layouts. Compared with the original 2D layouts, E_{flit} is reduced by up to 43.3 percent for Fat Trees and 30.7 percent for the Fat H-Tree. Thus, the 3D layouts of trees introduced here can shorten the wire length and improve the energy efficiency.

Note that the 3D layouts of mesh and torus in these graphs stand for the 3D mesh and torus. They exhibit better energy efficiency, since they can reduce the average hop count compared with their 2D forms, though their hardware amounts increase due to the upper and lower channels newly added for their vertical connections.

8.4 Leakage Power

Leakage power has already become a major component of the power consumption in recent process technologies. Assuming that the router is operating at 500 MHz, for example, the ratio of leakage in the total power of the router is 38 percent when no channel processes packets, though it is reduced down to 7 percent when every channel processes

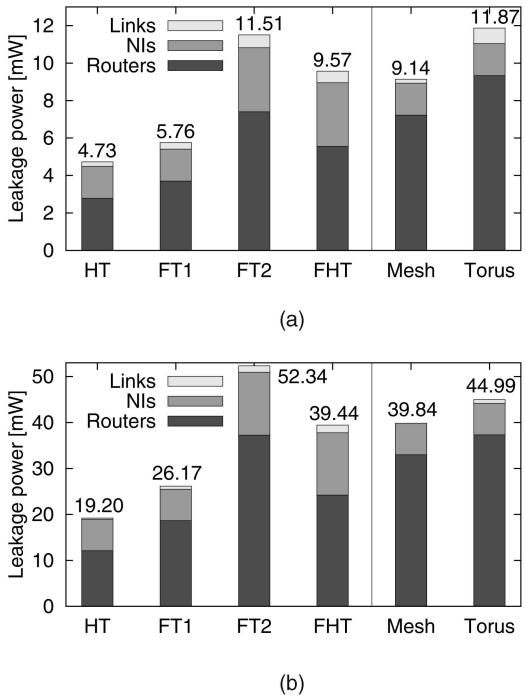

Fig. 14. Leakage power of 16- and 64-core typical networks. (a) 16-core network. (b) 64-core network.

a packet stream that utilizes 30 percent of the link bandwidth of a router link. Here, we estimated the total leakage power consumed in the network logic area of each topology. The NoC designs used in this experiment are the same as those used in Section 8.1. The supply voltage is 1.0 V, and the temperature is 25 °C. We also considered the leakage power of repeater buffers, assuming that a repeater is inserted every 1 mm on the network links.

Fig. 14 shows the total leakage power for the 16 and 64-core networks. Since the leakage power is proportional to the device area, the leakage power of Fat H-Tree is less than that of Fat Tree (2, 4, 2). It is comparable to that of a mesh, though it consumes more leakage in repeater buffers due to the long wire links. The Fat H-Tree has advantages in leakage power, and this impact will increase as the operation temperature increases or the process technology is scaled down.

8.5 Throughput

8.5.1 Simulation Environments

Network model. A flit-level network simulator written in C++ was used to confirm the deadlock freedom and measure the throughput on Fat H-Tree. A simple model corresponding to the wormhole router mentioned in Section 8.1 was used as a switching fabric in the simulator. A header flit requires at least three clock cycles to be transferred to the next router or core: one cycle for the routing computation, one cycle for allocating a virtual channel and a crossbar, and the remaining cycle for transferring the flit to the next router or core. Wormhole switching was used as a switching technique on the router. The nodes inject packets independently of each other, and we set the packet length for 16 flits including one header flit.

Routing algorithm. In this experiment, we focused on deterministic routing algorithms, since they can guarantee the in-order packet delivery with simple hardware logic compared with adaptive ones. MIN and TOR are used for Fat H-Tree. We omit the evaluation results of STR, since it gives twice the throughput of H-Tree while the others achieve more. Dimension-order routing (DOR) is used for a 2D mesh and torus. Since MIN and TOR in Fat H-Tree and DOR in torus require virtual channels to avoid structural deadlocks, two virtual channels are used for these 16 and 64-core networks.[3] The Fat H-Tree and Fat Trees provide alternative paths between their source and destination pairs. To generate deterministic routing path sets, we employ the path selection algorithm proposed in [37], which selects a single path from alternative paths so as to distribute the congestion over the network.

Traffic pattern. We used uniform synthesis traffic as a baseline. In addition, we used application traces captured from the NAS Parallel Benchmark (NPB) [38] programs, because they would enable us to evaluate using various sizes/patterns of real application traffic. We selected five matrix computation programs from NPB: Block Tridiagonal solver (BT), Scalar Pentadiagonal solver (SP), Conjugate Gradient (CG), Multi-Grid solver (MG), and large Integer Sort (IS). The class of problem is "W," and the numbers of tasks are 16 and 64. The task mapping that assigns each task into a core is a crucial factor for the average hop count and performance; thus, it was optimized for every trace in every network so that the pairs of tasks that transferred a large amount of data could be placed near each other. We employed the branch-and-bound method as a pruning technique to obtain the optimum mapping within a realistic time frame.

8.5.2 Simulation Results

First, we will present the performance advantages of Fat H-Tree over H-Tree and Fat Trees. Then, we will experimentally compare the Fat H-Tree with a 2D mesh and torus.

Fig. 15a shows the throughput (accepted traffic) versus the latency with the 16-core uniform traffic on Fat H-Tree and those on simple trees. In the graph, "FHTM" and "FHTT" represent the MIN and TOR of Fat H-Tree, respectively. The average hop count of each topology is also shown in parentheses (they are equal to the ones in Table 2). A Fat H-Tree (both cases of FHTM and FHTT) achieves an 11.1 percent shorter average hop count compared to Fat Trees. As for the throughput, a Fat H-Tree achieves a higher throughput than the others. As shown, the simulated throughput of FHTT outperforms that of Fat Tree (2, 4, 2) by 19.5 percent.

Similar results are also shown in the NPB traces. Figs. 15b, 15c, 15d, 15e, and 15f show the results for 16-core BT, SP, CG, MG, and IS traces. In these graphs, the Fat H-Tree outperforms the Fat Tree (2, 4, 2) in terms of the throughput and average hop count. Since the Fat H-Tree requires 19.8 percent-27.8 percent smaller network logic area compared to the Fat Tree, as reported in Section 8.1, it is clear that it outperforms the Fat Tree in terms of cost and performance. Thus, we can see that the Fat H-Tree is an attractive alternative to the conventional Fat Trees.

Next, we compare FHTM and FHTT. Although both FHTM and FHTT achieve higher throughput than Fat Trees as shown in Fig. 15a, 15b, 15c, 15d, 15e, and 15f, an interesting

3. TOR originally requires three virtual channels in a 64-core Fat H-Tree. To suppress the required virtual-channel number to two, torus-based paths that require three virtual channels are replaced with other paths that use only two, as mentioned in Section 5.

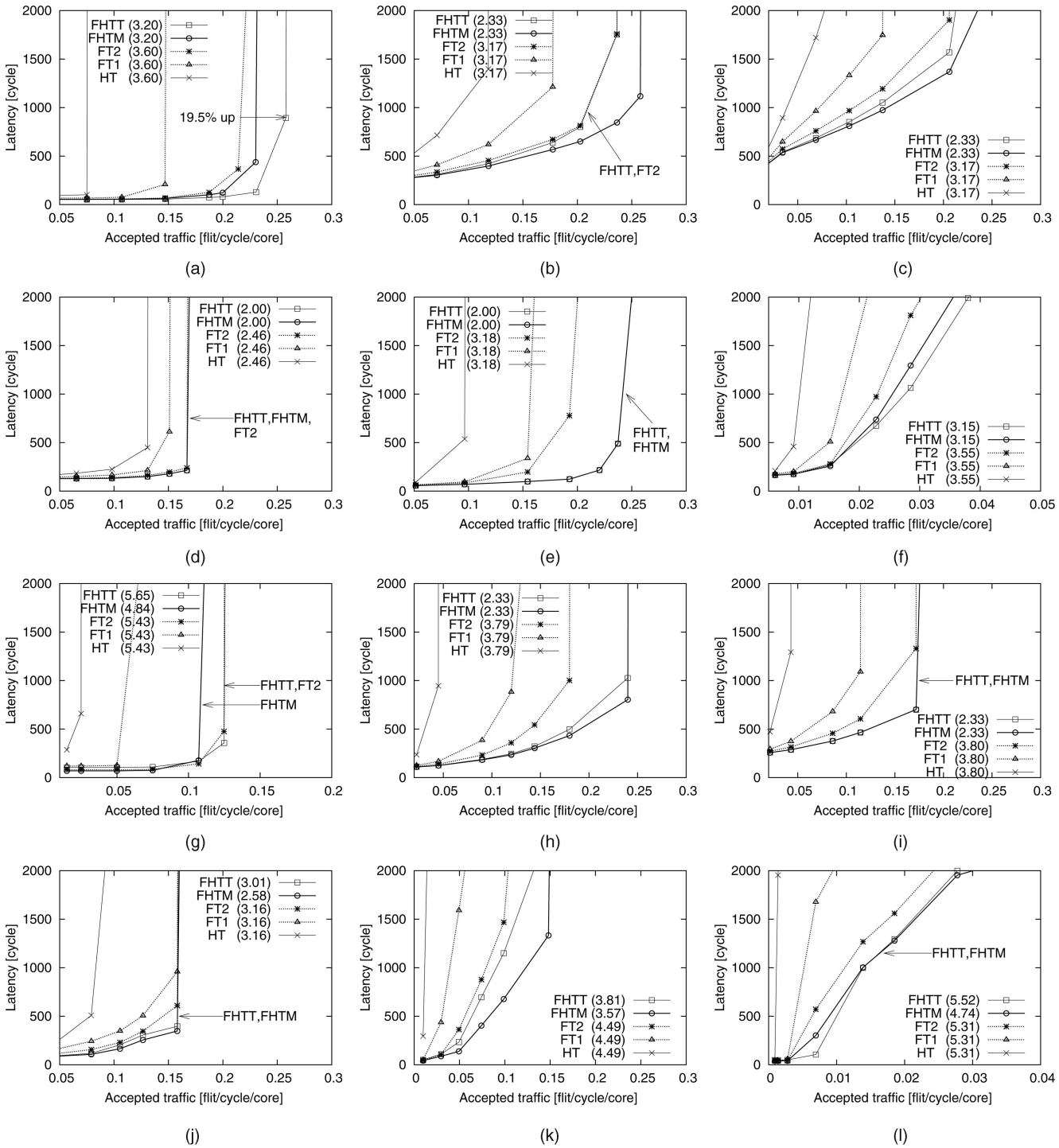

Fig. 15. Fat H-Tree versus tree-based topologies (H-Tree and Fat Trees). (a) Uniform (16-core). (b) BT.W (16-core). (c) SP.W (16-core). (d) CG.W (16-core). (e) MG.W (16-core). (f) IS.W (16-core). (g) uniform (64-core). (h) BT.W (64-core). (i) SP.W (64-core). (j) CG.W (64-core). (k) MG.W (64-core). (l) IS.W (64-core).

tendency can be seen in their performance differences. That is, FHTM outperforms FHTT in the BT.W and SP.W traces in which most communications are limited between neighboring cores. On the other hand, FHTT outperforms FHTM in the uniform traffic and IS.W trace, both of which are dominated by all-to-all communications. FHTM is able to exploit the links around the roots of a Fat H-Tree in order to always take minimal paths, and this strategy works well for most real application traces. However, for all-to-all communications, FHTM has no other choice but to use the links around the roots for taking minimal paths. As a result, FHTM causes the congestion around the roots. In such cases, FHTT is more suitable, since it is completely free from the root bottleneck, though it increases the hop count.

The simulation results with 64-core networks are shown in Figs. 15g, 15h, 15i, 15j, 15k, and 15l. As you can see, FHTM and FHTT achieve higher throughputs than the Fat Tree (2, 4, 2) in most traces.

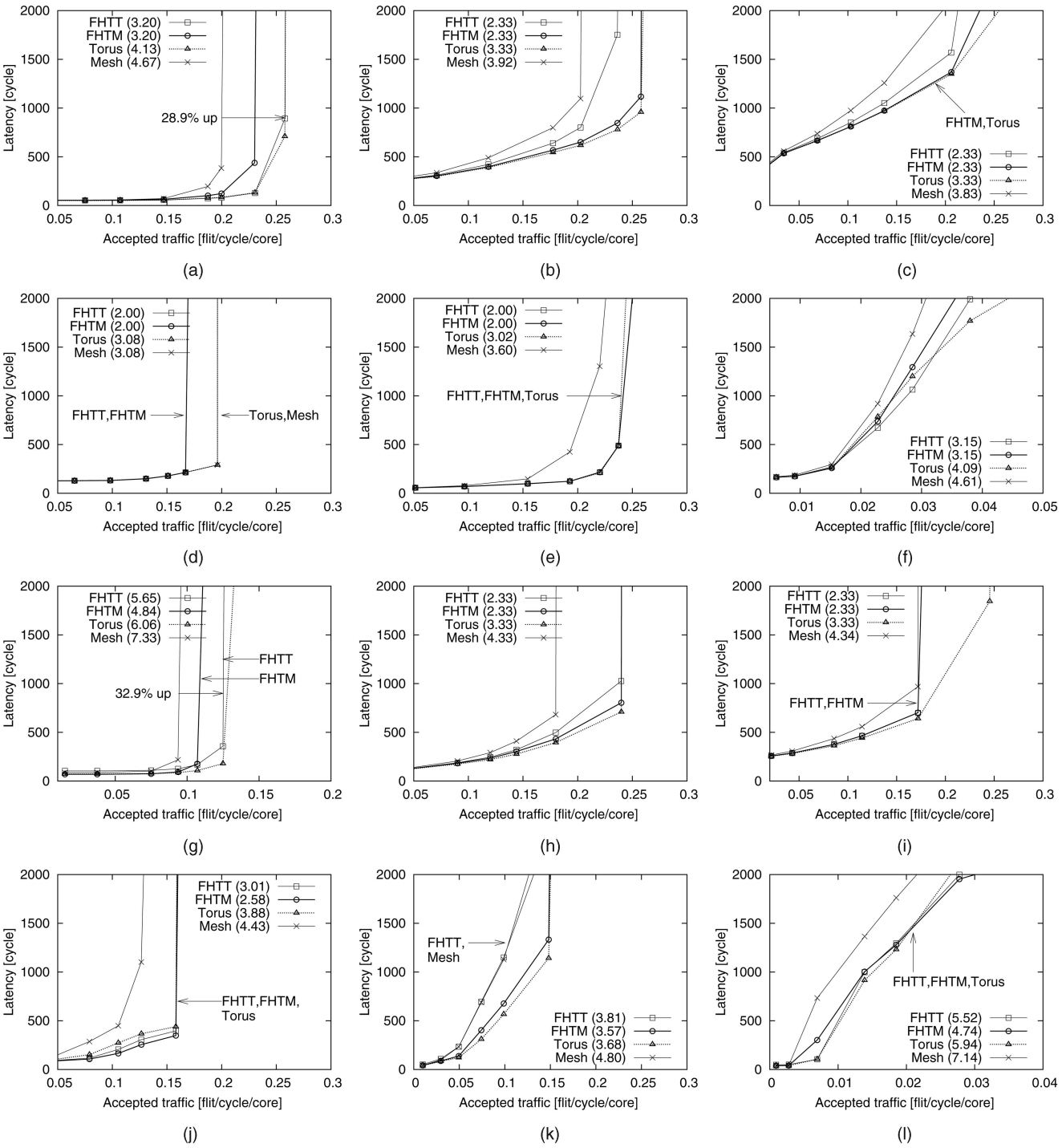

Fig. 16. Fat H-Tree versus grid-based topologies (2D mesh and torus). (a) Uniform (16-core). (b) BT.W (16-core). (c) SP.W (16-core). (d) CG.W (16-core). (e) MG.W (16-core). (f) IS.W (16-core). (g) Uniform (64-core). (h) BT.W (64-core). (i) SP.W (64-core). (j) CG.W (64-core). (k) MG.W (64-core). (l) IS.W (64-core).

Finally, the Fat H-Tree is compared with a 2D mesh and torus (Figs. 16a, 16b 16c, 16d, 16e, 16f, 16g, 16h, 16i, 16j, 16k, and 16l). The Fat H-Tree outperforms the mesh in all traces except for the 16-core CG.W, since the communication pattern of the 16-core CG.W is simple and can be optimized for mesh by using our task mapping technique. As shown in Figs. 16a and 16g, the simulated throughput of FHTT outperforms that of mesh by 28.9 percent and 32.9 percent in 16-core and 64-core uniform traffic, respectively. Since the required silicon budget for a Fat H-Tree is equal to or smaller than those for a mesh and torus, the Fat H-Tree could be comparable to these simple grid-based networks.

9 Conclusions

In this paper, we introduced the Fat H-Tree and its deadlock-free routing schemes. The 2D and 3D layout schemes of Fat H-Tree were also proposed. We showed

their properties and evaluated them. The evaluation results provided us with the following conclusions:

1. A Fat H-Tree outperforms a Fat Tree with two upward and four downward connections in terms of the throughput and average hop count.
2. A Fat H-Tree requires 19.8 percent-27.8 percent smaller network logic area than the Fat Tree.
3. A Fat H-Tree consumes 4.3 percent-5.0 percent less energy than the Fat Tree does.
4. A Fat H-Tree uses slightly more wire resources than the Fat Tree, but current process technology can provide sufficient wire resources for implementing Fat-H-Tree-based NoCs.
5. The 3D layout scheme reduces up to 49.8 percent of the wire resources of Fat H-Tree compared with its original 2D layout.

The Fat H-Tree achieves a higher performance compared to the Fat Tree, yet the required silicon budget for it is smaller than that for the Fat Tree; therefore, the Fat H-Tree topology proposed in this paper outperforms the conventional Fat Trees in terms of the cost and performance.

ACKNOWLEDGMENTS

A part of this work was supported by NII and JST CREST. The authors would like to thank the VLSI Design and Education Center and Kyoto University for a design flow of the ASPLA/STARC 90-nm CMOS process. When the work was done, Y. Yamada was with the Department of Information and Computer Science, Faculty of Science and Technology, Keio University.

REFERENCES

[1] W.J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," *Proc. 37th Design Automation Conf. (DAC '01),* pp. 684-689, June 2001.

[2] L. Benini and G.D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer,* vol. 35, no. 1, pp. 70-78, Jan. 2002.

[3] L. Benini and G.D. Micheli, *Networks on Chips: Technology and Tools.* Morgan Kaufmann, 2006.

[4] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J.F. Brown III, and A. Agarwal, "On-Chip Interconnection Architecture of the Tile Processor," *IEEE Micro,* vol. 27, no. 5, pp. 15-31, Sept. 2007.

[5] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S.W. Keckler, and D. Burger, "On-Chip Interconnection Networks of the TRIPS Chip," *IEEE Micro,* vol. 27, no. 5, pp. 41-50, Sept. 2007.

[6] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro,* vol. 27, no. 5, pp. 51-61, Sept. 2007.

[7] C.E. Leiserson, "Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing," *IEEE Trans. Computers,* vol. 34, no. 10, pp. 892-901, Oct. 1985.

[8] A. DeHon, "Compact, Multilayer Layout for Butterfly Fat-Tree," *Proc. 12th Ann. ACM Symp. Parallel Algorithms and Architectures (SPAA '00),* pp. 206-215, July 2000.

[9] A. DeHon, "Unifying Mesh- and Tree-Based Programmable Interconnect," *IEEE Trans. Very Large Scale Integration Systems,* vol. 12, no. 10, pp. 1051-1065, Oct. 2004.

[10] A. Andriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C.A. Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network," *Proc. Design Automation and Test in Europe Conf. (DATE '03),* pp. 70-73, Mar. 2003.

[11] C. Grecu, P.P. Pande, A. Ivanov, and R. Saleh, "Structured Interconnect Architecture: A Solution for the Non-Scalability of Bus-Based SoCs," *Proc. 14th ACM Great Lakes Symp. VLSI (GLSVLSI '04),* pp. 192-195, Apr. 2004.

[12] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M.J. Wilson, M. Wrighton, and A. DeHon, "Packet-Switched vs. Time-Multiplexed FPGA Overlay Networks," *Proc. 14th Ann. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM '06),* pp. 205-216, Apr. 2006.

[13] S. Das, A. Fan, K.-N. Chen, C.S. Tan, N. Checka, and R. Reif, "Technology, Performance, and Computer-Aided Design of Three-Dimensional Integrated Circuits," *Proc. Int'l Symp. Physical Design (ISPD '04),* pp. 108-115, Apr. 2004.

[14] W.R. Davis, J. Wilson, S. Mick, J. Xu, H. Hua, C. Mineo, A.M. Sule, M. Steer, and P.D. Franzon, "Demystifying 3D ICs: The Pros and Cons of Going Vertical," *IEEE Design and Test of Computers,* vol. 22, no. 6, pp. 498-510, Nov. 2005.

[15] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G.H. Loh, D. McCaule, P. Morrow, D.W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J.P. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," *Proc. 39th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO '06),* pp. 469-479, Dec. 2006.

[16] C.E. Leiserson, Z.S. Abuhamdeh, D.C. Douglas, C.R. Feynman, M.N. Ganmukhi, J.V. Hill, W.D. Hillis, B.C. Kuszmaul, M.A.S. Pierre, D.S. Wells, M.C. Wong-Chan, S.-W. Yang, and R. Zak, "The Network Architecture of the Connection Machine CM-5," *J. Parallel and Distributed Computing,* vol. 33, no. 2, pp. 145-158, Mar. 1996.

[17] Y. Yang, A. Funahashi, A. Jouraku, H. Nishi, H. Amano, and T. Sueyoshi, "Recursive Diagonal Torus: An Interconnection Network for Massively Parallel Computers," *IEEE Trans. Parallel and Distributed Systems,* vol. 12, no. 7, pp. 701-715, July 2001.

[18] F.T. Leighton, "New Lower Bound Techniques for VLSI," *Math. Systems Theory,* vol. 17, no. 1, pp. 47-70, Apr. 1984.

[19] W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann, 2004.

[20] J. Kim, W.J. Dally, B. Towles, and A.K. Gupta, "Microarchitecture of a High-radix Router," *Proc. 32nd Int'l Symp. Computer Architecture (ISCA '05),* pp. 420-431, June 2005.

[21] J. Kim, J. Balfour, and W.J. Dally, "Flattened Butterfly Topology for On-Chip Networks," *Proc. 40th Ann. IEEE/ACM Int'l Symp. Microarchitecture (MICRO '07),* pp. 172-182, Dec. 2007.

[22] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: A Novel On-Chip Communication Network," *Proc. Int'l Symp. System-on-Chip (ISSOC '04),* p. 15, Nov. 2004.

[23] R. Sabbaghi-Nadooshan, M. Modarressi, and H. Sarbazi-Azad, "A Novel High-Performance and Low-Power Mesh-Based NoC," *Proc. Int'l Workshop Performance Modeling, Evaluation, and Optimization of Ubiquitous Computing and Networked Systems (PMEO-UCNS '08),* Apr. 2008.

[24] A. Sharifi, R. Sabbaghi-Nadooshan, and H. Sarbazi-Azad, "The Shuffle-Exchange Mesh Topology for 3D NoCs," *Proc. Ninth Int'l Symp. Parallel Architectures, Algorithms, and Networks (I-SPAN '08),* pp. 275-280, May 2008.

[25] G.D. Vecchia and C. Sanges, "A Recursively Scalable Network VLSI Implementation," *Future Generation Computer Systems,* vol. 4, no. 3, pp. 235-243, Oct. 1988.

[26] D. Rahmati, A.E. Kiasari, S. Hessabi, and H. Sarbazi-Azad, "A Performance and Power Analysis of WK-Recursive and Mesh Networks for Network-on-Chips," *Proc. 24th Int'l Conf. Computer Design (ICCD '06),* pp. 142-147, Oct. 2006.

[27] Y. Yamada, H. Amano, M. Koibuchi, A. Jouraku, K. Anjo, and K. Nishimura, "Folded Fat H-Tree: An Interconnection Topology for Dynamically Reconfigurable Processor Array," *Proc. Int'l Conf. Embedded and Ubiquitous Computing (EUC '04),* pp. 301-311, Aug. 2004.

[28] H. Matsutani, M. Koibuchi, and H. Amano, "Performance, Cost, and Energy Evaluation of Fat H-Tree: A Cost-Efficient Tree-Based On-Chip Network," *Proc. 21st Int'l Parallel and Distributed Processing Symp. (IPDPS '07),* Mar. 2007.

[29] H. Matsutani, M. Koibuchi, D.F. Hsu, and H. Amano, "Three-Dimensional Layout of On-Chip Tree-Based Networks," *Proc. Ninth Int'l Symp. Parallel Architectures, Algorithms, and Networks (I-SPAN '08),* pp. 281-288, May 2008.

[30] T.M. Pinkston and J. Shin, "Trends toward On-Chip Networked Microsystems," *Int'l J. High Performance Computing and Networking,* vol. 3, no. 1, pp. 3-18, Sept. 2005.

[31] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory," *Proc. 33rd Int'l Symp. Computer Architecture (ISCA '06),* pp. 130-141, June 2006.
[32] W.J. Dally and J.W. Poulton, *Digital Systems Eng.* Cambridge Univ. Press, 1998.
[33] A. Banerjee, R. Mullins, and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architectures," *Proc. First Int'l Symp. Networks-on-Chip (NOCS '07),* pp. 163-172, May 2007.
[34] H. Matsutani, M. Koibuchi, D. Wang, and H. Amano, "Adding Slow-Silent Virtual Channels for Low-Power On-Chip Networks," *Proc. Second Int'l Symp. Networks-on-Chip (NOCS '08),* pp. 23-32, Apr. 2008.
[35] H. Wang, L.-S. Peh, and S. Malik, "A Technology-Aware and Energy-Oriented Topology Exploration for On-Chip Networks," *Proc. Design, Automation and Test in Europe Conf. (DATE '05),* pp. 1238-1243, Mar. 2005.
[36] R. Ho, K.W. Mai, and M.A. Horowitz, "The Future of Wires," *Proc. IEEE,* vol. 89, no. 4, pp. 490-504, Apr. 2001.
[37] J.C. Sancho and A. Robles, "Improving the Up*/Down* Routing Scheme for Networks of Workstations," *Proc. Int'l Euro-Par Conf. Parallel Processing (Euro-Par '00),* pp. 882-889, Aug. 2000.
[38] D. Bailey, T. Harris, W. Saphir, R. van der Wijngaart, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0," NAS Technical Report NAS-95-020, Dec. 1995.

Hiroki Matsutani received the BA, ME, and PhD degrees from Keio University, Yokohama, Japan, in 2004, 2006, and 2008, respectively. He is currently a visiting researcher in the Department of Information and Computer Science, Faculty of Science and Technology, Keio University and a cooperative researcher in the Research Center for Advanced Science and Technology (RCAST), University of Tokyo, Japan. He has been awarded a Research Fellowship of the Japan Society for the Promotion of Science (JSPS) for Young Scientists since 2006. His research interests include the areas of networks on chips and interconnection networks. He is a student member of the IEEE.

Michihiro Koibuchi received the BE, ME, and PhD degrees from Keio University, Japan, in 2000, 2002, and 2003, respectively. He was a visiting researcher at the Technical University of Valencia, Spain, in 2004 and a visiting scholar at the University of Southern California, in 2006. He is currently an assistant professor in the Infrastructure Systems Research Division, National Institute of Informatics, Tokyo, and the Graduate University for Advanced Studies, Japan. His research interests include the areas of high-performance computing and interconnection networks. He is a member of the IEEE.

Yutaka Yamada received the BE and ME degrees from Keio University, Japan, in 2003 and 2005, respectively. He is currently working as a research engineer in the Research and Development Center, Toshiba Corporation, Japan. His research interests include the areas of reconfigurable systems.

D. Frank Hsu received the BS degree from Cheng Kung University, Tainan, Taiwan, the MS degree from the University of Texas, El Paso, and the PhD degree from the University of Michigan. He is the Clavius professor of science and a professor of computer and information science in the Department of Computer and Information Sciences and the associate dean of the Graduate School of Arts and Sciences, Fordham University, New York. He has held visiting positions at the Massachusetts Institute of Technology, Boston University, Keio University (as the IBM chair professor), the Japan Advanced Institute of Science and Technology (as the Komatsu chair professor), Taiwan University, Tsing Hua University, Hsinchu, Taiwan, and the University of Paris-Sud, France. He has also served as the chair of the Section of Computer and Information Science, New York Academy of Sciences. His research areas are combinatorics and graph theory, computational sciences and intelligence, network interconnections and communications, and informatics and intelligent systems. His recent work on combinatorial fusion analysis (CFA) has applications in biomedicine, virtual screening and drug discovery, target recognition and tracking, information and music retrieval, and cyber security. He has served on several committees of international conferences including three DIMACS workshops, the IEEE-AINA conference series, the I-SPAN conference series, and the EITC-Bioinformatics, Biomedicine, and Biotechnology (EITC-Bio) conference series. He has served on the editorial board of several journals, including *IEEE Transactions on Computers*, *Networks*, *International Journal of Foundation of Computer Science*, and *Monograph on Combinatorial Optimization*. He is an associate editor of *Pattern Recognition Letters* and has served as the editor in chief for special issues of the *Journal of Interconnection Networks*. He is the recipient of the Bene Merenti Meritorious Service Award and a Distinguished Teaching Award. He is a senior member of the IEEE and the IEEE Computer Society, a Foundation Fellow of the Institute of Combinatorics and Applications (TICA), and a fellow of the New York Academy of Sciences.

Hideharu Amano received the PhD degree from Keio University, Yokohama, Japan, in 1986. He is currently a professor in the Department of Information and Computer Science, Faculty of Science and Technology, Keio University. His research interests include the areas of parallel processing and reconfigurable systems. He is a member of the IEEE.